<http://www.java2novice.com/java_interview_questions/thread-race-condition/>

MultiThreading

Race Condition

A race condition is a situation in which two or more threads orprocesses are reading or writing some shared data, and the finalresult depends on the timing of how the threads are scheduled.Race onditions can lead to unpredictable results and subtleprogram bugs. A thread can prevent this from happening by lockingan object. When an object is locked by one thread and anotherthread tries to all a synchronized method on the same object,the second thread will be block until the object is unlocked.

The situation where multiple threads try to operate on a shared resource without proper synchronization and sequence of their operations interleave is **Race Condition**( Also see [Thread interference and Memory consistency errors](http://javasolutionsonline.blogspot.in/2013/09/thread-interference-and-memory.html)) .  
  
**Example 1 .**  
The most basic example to describe a Race condition is a Counter class that increments value of an instance variable .

[viewplaincopy to clipboardprint?](http://javasolutionsonline.blogspot.in/2013/09/race-condition-in-concurrency.html)

1. **class** Counter {
2. **privateint** c = 0;
3. **publicvoid** increment() {
4. c++;
5. }
6. }

class Counter {

private int c = 0;

public void increment() {

c++;

}

}

**Note :**  
It seems that the increment operation is atomic ( Atomic operations can't be interleaved by threads ) but it is not . It can further be divided into three steps :  
**1.** Read the value c.   
**2.** Increment the value .  
**3.** Write the incremented value.  
  
  
Lets imagine that two threads A and B increments the value of instance variable c . Ideally if there is no interleaving of thread's operations , the value of the variable should be 2 as both the threads increment it by 1 . Lets see one of the possible scenario where thread's operations interleave   
  
Thread A : Read c ( which is 0 ).  
Thread B : Read c ( Still 0 ).  
Thread A : Increments the value by 1.  
Thread A : Write the value ( Now c is 1 )  
Threab B : Increments the value   
Thread B : Write the value ( Still c is 1 )   
  
Here we can see that due to interleaving of Read operations of two threads , the final value of the variable c is 1 and not 2.   
  
**Example 2 .**  
Below is a short snippet from a singleton class . Here first the code checks whether the instance is **null** and then creates a new object . The purpose of this code is to ensure that there is only one instance of **MySingleton**class.

[viewplaincopy to clipboardprint?](http://javasolutionsonline.blogspot.in/2013/09/race-condition-in-concurrency.html)

1. **public**MySingletongetInstance(){
2. **if** (instance == **null**){
3. instance = **new**MySingleton();
4. }
5. }

public MySingleton getInstance(){

if (instance == null){

instance = new MySingleton();

}

}

Now lets consider that there are two threads trying to get an instance of **MySingleton**class by calling the method getInstance() . It might be possible , due to interleaving of thread's operations , that both the thread sees the value of instance as **null** and in that case , two objects will get created by the threads .   
  
**How to fix the problem ?**  
Race conditions can be avoided by proper synchronization of the code . As an application developer , you have to enforce locking to make sure that only one thread enters a critical section at a time and the result of the thread is visible to others once it comes out of the synchronized block .

Spring Scoped Proxy bean:

<http://stackoverflow.com/questions/14371335/spring-scoped-proxy-bean>